BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ❑ BLACK BORDERS
- ❑ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- ❑ FADED TEXT OR DRAWING
- ❑ BLURRED OR ILLEGIBLE TEXT OR DRAWING
- ❑ SKEWED/SLANTED IMAGES
- ❑ COLOR OR BLACK AND WHITE PHOTOGRAPHS
- ❑ GRAY SCALE DOCUMENTS
- ❑ LINES OR MARKS ON ORIGINAL DOCUMENT
- ☒ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- ❑ OTHER: _____

IMAGES ARE BEST AVAILABLE COPY.
As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.

PUB. NO.: 09-311786 [JP 9311786 A]
PUBLISHED: December 02, 1997 (19971202)
INVENTOR(s): UMINAGA MASAHIRO
SAITO YASUHIKO
APPLICANT(s): HITACHI LTD [000510] (A Japanese Company or Corporation), JP (Japan)
APPL. NO.: 09-052772 [JP 9752772]
FILED: March 07, 1997 (19970307)
INTL CLASS: [6] G06F-009/38
JAPIO CLASS: 45.1 (INFORMATION PROCESSING -- Arithmetic Sequence Units)
JAPIO KEYWORD:R131 (INFORMATION PROCESSING -- Microcomputers & Microprocessers)

ABSTRACT

PROBLEM TO BE SOLVED: To reduce a pipeline stall due to a data hazard of a superscalar system and to improve the processing speed by changing an instruction in 1st instruction format stored in an instruction memory into an instruction in 2nd instruction format.

SOLUTION: The instruction is taken in a 1st stage from the instruction memory and the instruction taken in the 1st stage 101 is decoded in a 2nd stage 103. The decoded instruction is executed in a 3rd stage and when the execution result is written in a register in a 4th stage 107, the instruction in the 1st instruction format stored in the instruction memory is changed into the instruction in the 2nd instruction format and executed. Consequently, the pipeline stall due to the data hazard of the superscalar system can be reduced and the processing speed is improved.

(19)日本国特許庁（ＪＰ）　(12) 公開特許公報（A）　(11)特許出願公開番号

特開平9－311786

(43)公開日　平成９年(1997)12月２日

(51)Int.Cl.[6]	識別記号	庁内整理番号	ＦＩ		技術表示箇所
Ｇ０６Ｆ　9/38	３１０		Ｇ０６Ｆ　9/38	３１０Ｘ	

審査請求　未請求　請求項の数20　ＯＬ　（全 17 頁）

(21)出願番号　特願平9－52772

(22)出願日　平成９年(1997)３月７日

(31)優先権主張番号　特願平8－60571

(32)優先日　平８(1996)３月18日

(33)優先権主張国　日本（ＪＰ）

(71)出願人　000005108
株式会社日立製作所
東京都千代田区神田駿河台四丁目６番地

(72)発明者　海永　正博
東京都小平市上水本町五丁目20番１号　株式会社日立製作所半導体事業部内

(72)発明者　斎藤　靖彦
東京都小平市上水本町五丁目20番１号　株式会社日立製作所半導体事業部内

(74)代理人　弁理士　小川　勝男

(54)【発明の名称】　データ処理装置

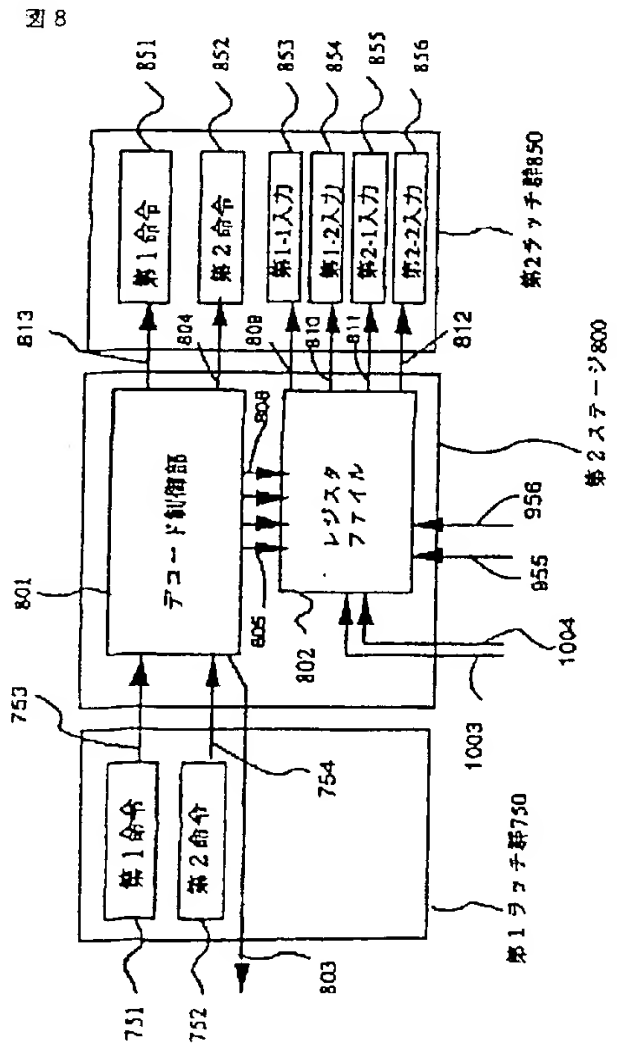
(57)【要約】

【課題】　スーパスカラ方式におけるデータ・ハザードのよるパイプライン・ストールを削減し、処理速度の向上を実現することにある。

【解決手段】　隣接する２つの２オペランド命令が、１つの３オペランド命令と同等であることを検出する回路と、そうであれば２つの命令を１つの３オペランド命令に統合して後続の実行ステージに送出する回路を命令デコーダに設ける。また隣接する２つの命令がデータフローの関係にあるが１つの３オペランド命令には統合できないことを検出すると、先行命令のソースデータを後続命令のための演算器に送る回路を設ける。

【効果】　隣接命令間のデータフローにより従来であれば２クロックの時間を要していた２つの命令処理を１クロックで実行できる。したがって、全体としての実行クロック数を削減できる。

図８



【特許請求の範囲】

【請求項１】複数のステージに分割して命令を実行するデータ処理装置であって、

前記複数のステージは、少なくとも命令メモリから命令を取り込む第１のステージと、前記第１のステージで取り込んだ命令を解読する第２のステージと、前記第２のステージで解読された命令を実行する第３のステージと、前記第３のステージで実行された結果をレジスタに書き込む第４のステージとであり、

前記命令メモリに格納される第１の命令フォーマットの命令を第２の命令フォーマットの命令に変更して実行することを特徴とするデータ処理装置。

【請求項２】前記第１の命令フォーマットは、演算命令において第１のオペランドと第２のオペランドとを演算し、第２のオペランドに演算結果を格納する命令フォーマットであり、

前記第２の命令フォーマットは、演算命令において第１のオペランドと第２のオペランドとを演算し、第３のオペランドに演算結果を格納する命令フォーマットであることを特徴とする請求項１に記載のデータ処理装置。

【請求項３】前記第２のステージは、先行命令がレジスタ間のデータ転送命令であることを検出し、後続命令が演算命令であることを検出し、さらに先行命令の転送先レジスタ番号と後続命令の転送先レジスタ番号が同一であることを検出して、

前記第２の命令フォーマットの演算命令に変換して前記第３のステージに送出することを特徴とする請求項２に記載のデータ処理装置。

【請求項４】前記請求項３に記載のデータ処理装置は、単一の半導体基板上に形成される。

【請求項５】前記先行命令が転送元レジスタの内容をそのまま転送先レジスタに転送するデータ転送命令である請求項４に記載のデータ処理装置。

【請求項６】前記先行命令が転送先のレジスタの内容をシフトして転送先のレジスタに転送するデータ転送命令である請求項４に記載のデータ処理装置。

【請求項７】前記先行命令が転送元レジスタの内容を０拡張または符号拡張して転送元のレジスタに転送するデータ転送命令である請求項４に記載のデータ処理装置。

【請求項８】前記第２の命令フォーマットは、前記第１の命令フォーマットの命令を複数組み合わせた命令を有することを特徴とする請求項１に記載のデータ処理装置。

【請求項９】前記第２のステージは、先行命令がレジスタ間のデータ転送であることを検出し、後続命令が固定ビットシフト命令であることを検出し、さらに先行命令の転送先レジスタ番号と後続命令の転送元レジスタ番号が同一であることを検出して、

前記第２の命令フォーマットである１つのシフト命令に変換して第３のステージに送出することを特徴とする請求項８に記載のデータ処理装置。

【請求項１０】前記第２ステージは、先行命令がレジスタ間のデータ転送命令であることを検出し、後続命令が演算命令であることを検出し、さらに先行命令の転送先レジスタ番号と後続命令の転送元レジスタ番号が同一であることを検出して、

後続命令を先行命令とデータフローの関係にない前記第２の命令フォーマットの演算命令に変換して前記第３のステージに送出し、複数の同一ステージを並列実行可能とすることを特徴とする請求項２に記載のデータ処理装置。

【請求項１１】前記第１の命令フォーマットは、２バイト固定長命令であることを特徴とする請求項１０に記載のデータ処理装置。

【請求項１２】前記先行命令が転送元レジスタの内容をそのまま転送先レジスタに転送するデータ転送命令である請求項１１に記載のデータ処理装置。

【請求項１３】前記先行命令が転送先のレジスタの内容をシフトして転送先のレジスタに転送するデータ転送命令である請求項１１に記載のデータ処理装置。

【請求項１４】前記先行命令が転送元レジスタの内容を０拡張または符号拡張して転送元のレジスタに転送するデータ転送命令である請求項１１に記載のデータ処理装置。

【請求項１５】パイプライン方式のデータ処理装置であって、

命令メモリに格納される固定長命令を読み込む第１のステージと、

読み込まれた複数の命令が実行するデータに依存性が有り、かつ前記複数の命令に所定の関係がある場合、前記複数の命令を複数のパイプラインで並列に実行できるように前記複数の命令を変更する第２のステージと、

変更された前記複数の命令を並列に実行する第３のステージとを有することを特徴とするデータ処理装置。

【請求項１６】請求項１５に記載の第１のステージは２つの命令を同時に読み込み、第２のステージは２つの命令を２本のパイプラインで並列に実行できるように前記２つの命令を変更することを特徴とするデータ処理装置。

【請求項１７】請求項１６に記載の第１のステージは、２バイト固定長命令を読み込むことを特徴とするデータ処理装置。

【請求項１８】ＣＰＵと命令メモリとを単一の半導体基板上に形成するマイクロコンピュータであって、

前記ＣＰＵは、

命令メモリに格納される２バイト固定長命令を２つ読み込む命令フェッチユニットと、

読み込まれた前記２つの命令が実行するデータに依存性が有り、かつ前記２つの命令に所定の関係がある場合、２つの命令を２本のパイプラインで並列に実行できるよ

うに前記２つの命令を変更する命令デコーダと、
変更された２つの命令を並列に実行する２つの４バイト長の演算器とを有することを特徴とするマイクロコンピュータ。
【請求項１９】請求項１８に記載の前記命令デコーダは、演算命令において第１のオペランドと第２のオペランドとを演算し、
第２のオペランドに演算結果を格納する命令を、第１のオペランドと第２のオペランドとを演算し、第３のオペランドに演算結果を格納する命令に変更することを特徴とするマイクロコンピュータ。
【請求項２０】請求項１８に記載の前記命令デコーダは、先行命令がレジスタ間のデータ転送命令であることを検出し、後続命令が演算命令であることを検出し、
さらに先行命令の転送先レジスタ番号と後続命令の転送元レジスタ番号が同一であることを検出して、後続命令を先行命令とデータフローの関係にない演算命令に変更することを特徴とするマイクロコンピュータ。
【発明の詳細な説明】
【０００１】
【発明の属する技術分野】本発明は、マイクロプロセッサやマイクロコンピュータ等のデータ処理装置に係わり、特にスーパスカラ等の並列処理を行うデータ処理装置に適用して有効な技術に関する。
【０００２】
【従来の技術】マイクロプロセッサ（ＣＰＵ(Central Processing Unit)、マイクロコンピュータ等の総称として以下使用する。）は、命令の列を順次にフェッチし、解読し、実行していく。マイクロプロセッサが実行していく命令は、デコード回路の簡単化を狙って現在固定長のものが広まってきている。固定長命令をパイプライン方式(Pipelining)で実行するマイクロプロセッサは、ＲＩＳＣ(Redused Instruction Set Computer)型プロセッサと呼ばれている。
【０００３】図１は、マイクロプロセッサのパイプライン化された実現方法を示したものである。ここでは簡単化のため通常は存在するメモリアクセスのステージ（MEM）を省略してある。個別のステージ（１０１、１０３、１０５、１０７）は１単位の時間刻み（クロック）で実行され、最初のステージから最後のステージまでラッチ群（１０２、１０４、１０６）を介して順次に処理を積み重ねていくことで個別の命令処理が完了する。第１ステージ１０１は命令フェッチを行う（ＩＦ）。第２ステージ１０３は命令の解釈及びレジスタの読み出しを行う（ＩＤ）。第３ステージ１０５は命令機能が指定した演算を実行する（ＥＸ）。第４ステージ１０７は演算結果を信号線１０８を介して第２ステージ１０３内に配置されたレジスタに書き込みを行う（ＷＢ）。
【０００４】図２にはパイプラインで４つの命令を処理していく時の概念図が示される。後続の命令が先行命令のレジスタの内容を使用する場合は、後続の命令のパイプラインに空きができてしまう(データ・ハザードによるパイプライン・ストールと呼ばれる)。この様子が図２の（ａ）に示されている。図２の（ａ）内の左下を向いた２つの矢印は先行命令のレジスタ書き込み後、後続命令のレジスタ読み出しを示している。
【０００５】したがって、この問題を解決する手段として、後続の命令が前の演算結果を使用する場合にはその値を信号線１０８を介して第３ステージ１０５内の演算器にも送出する。以上のための制御線が信号線１０９、１１０である。この調整はフォワーディング(Forwarding)として知られており、これにより１クロック毎の実行が可能となる。なお、図２の（ｂ）内の左下を向いた２つの矢印はフォワーディングを示している。したがって個別の命令処理に要するクロック数は例えば４となる。しかし、個別のステージが毎クロック新たな命令を処理していくので、命令処理は１クロック当たり１命令となる。したがって、１命令が１クロックで実行できるので、ある処理(プログラム)を行うための実行命令数が少ないほど実行時間が短くなる。
【０００６】なお、パイプライン及びフォワーディングについては、１９９４年Morgan Kaufman Publishers, Inc. 発行のHennessy et al. 「Computer Organization and Design」第６章Enhancing Performance with Pipelining(３６２頁から４５０頁)に記載されている。
【０００７】次に、マイクロプロセッサの処理速度を向上する方式の１例として、スーパスカラ方式(Superscalar)がある。スーパスカラ方式は、同時に実行できる演算器の数を複数個、例えば２個にし、それに応じて命令フェッチと命令デコードも１時期に２つ行えるようにしたものである。この場合、図３の（ａ）データ依存無しに示されるように、理想的には１クロック毎に２つの命令が実行可能にされるので、通常のパイプライン方式に較べ実行時間が半分になる。なお、スーパスカラ方式については、日経エレクトロニクス、１９８９年１１月２７日号(No.487)、１９１頁から２００頁の「次世代ＲＩＳＣ、並列処理を導入しＣＭＯＳで１００ＭＩＰＳねらう」に記載されている。
【０００８】従来のスーパスカラ方式を採用しているＲＩＳＣ型のマイクロプロセッサの命令長は４バイト固定であり、算術演算等の演算命令のオペランド数は３つとなっているのが一般的である。この例は、特開平２－１３０６３４号に記載されている。一方、コード効率を高める（命令を格納するメモリの使用量を少なくする）ために、２バイト固定長命令のＲＩＳＣ型のマイクロプロセッサがある。ただし、前記２バイト固定長命令のＲＩＳＣ型のマイクロプロセッサにはスーパスカラ方式は採用されていない。この例は、特開平５－１９７５４６号に記載されている。
【０００９】

【発明が解決しようとする課題】スーパスカラ方式により生ずる課題を明らかにするために、図３を用いて説明する。図３に示される命令の動作が下記に示される。

【００１０】（１）mov　R3, R2　「レジスタR3の内容をレジスタR2に複写」

（２）mov　#32, R5「データ*32*をレジスタR5に複写」

（３）add　R4, R2　「レジスタR4の内容とR2の内容を加算して、結果をR2に格納」

（４）and　R3, R5　「レジスタR3の内容とR5の内容を論理積して、結果をR5に格納」

上記命令（１）と命令（２）、及び命令（３）と命令（４）にはそれぞれデータの依存性（データフロー）はない。しかし、命令（１）と命令（３）、及び命令（２）と命令（４）にはそれぞれデータの依存性（データフロー）がある。すなわち、命令（１）と命令（３）の両方でレジスタR2を使用する。また、命令（２）と命令（４）の両方でレジスタR5を使用する。従って、命令（１）の実行後に命令（３）を実行しなければならない。また、命令（２）の実行後に命令（４）を実行しなければならない。

【００１１】すなわち、同時に実行する命令間にデータ依存性が無い場合、図３の（ａ）に示されるようにパイプラインの空きが無く、２命令が完全に並列実行されので、従来の同時に１命令のみを実行する場合の２倍の処理速度が得られる。しかし、同時に実行する命令間にデータ依存性がある場合、図３の（ｂ）に示されるように、パイプラインに乱れが出てしまい、従来の同時に１命令のみを実行する場合と同一の処理速度になってしまう。

【００１２】そのために、図３の（C）に示されるように、同時に実行する命令間にデータ依存性がある場合、後続命令は次のパイプラインに回し、後続命令の替りに無処理命令nopを先行命令と同時に実行して、パイプラインの乱れを回避する方法が考えられる。しかし、無駄な命令が増え、全体の実行命令数が増加して実行時間が長くなる。

【００１３】次に命令フォーマット及び命令体系により生ずる課題を明らかにするため、図４及び図５を用いて以下に説明する。

【００１４】図４には、４バイト・３オペランド命令（４バイト固定長命令）体系の場合の命令形式（命令フォーマット）と命令レパートリの例が示される。この図でＯＰフィールド４０１は命令機能を特定する。Ｓ１フィールド４０３は第１入力を特定するレジスタ番号（第１オペランド）、Ｓ２フィールド４０４は第２入力を特定するレジスタ番号（第２オペランド）、Ｄフィールド４０２は出力を特定するレジスタ番号（第３オペランド）が置かれている。すなわち、この命令形式は３つのオペランドを指定することができる。命令機能には、複写（データ転送）、加算、減算などがある。さらに、４バイト命令体系の命令長の余裕から１ビット左シフト加算命令asl1addや０拡張加算命令zextaddなどの複合命令も提供される。asl1add命令は第１オペランドのビットパタンを１ビット左シフトした後で通常の加算を行うもので、zextadd命令は第１オペランドのビットパタンの左半分を０にした後で通常の加算を行うものである。なおここでは簡単化のため通常は存在するであろうメモリアクセス命令や分岐命令等を省略してある。なお複写命令（データ転送命令）の場合Ｓ２フィールド４０４は無視され、Ｓ１フィールド４０３で特定されたレジスタ（転送元レジスタ）内容がそのままＤフィールド４０２で特定されたレジスタ（転送先レジスタ）に複写（転送）される。

【００１５】図５には、２バイト・２オペランド命令（２バイト固定長命令）体系の場合の命令形式と命令レパートリの例が示される。図５でＯＰフィールド５０１は命令機能を特定する。Ｓ１フィールド５０３は第１入力を特定するレジスタ番号（第１オペランド）、Ｄフィールド５０２は第２入力を特定するレジスタ番号（出力を特定するレジスタ番号と同一、第２オペランド）が置かれている。すなわち、この命令形式は２つのオペランドを指定することができる。図４と較べてフS２フィールドが存在しない点が図４の命令形式と明確に異なっている部分である。すなわち、オペランドの数が１つ少ない。さらに残りのフィールド長も図４のものに較べて短くなっている。

【００１６】命令機能には１入力転送命令として複写命令（データ転送命令）、０拡張命令、符号拡張命令、１ビット左シフト命令、２入力演算命令として加算命令、減算命令等がある。このうち１ビット左シフト命令は、命令長の都合で入力レジスタ（転送元レジスタ）と出力レジスタ（転送先レジスタ）の番号が同じである。したがってこの場合、S1フィールドはレジスタ番号でなく、asl1命令を特定するための拡張命令コードが格納される。

【００１７】さて、４バイト・３オペランド命令体系と２バイト・２オペランド命令体系の利害得失を明確化するために例えば、以下の式を考える。

【００１８】

（A）

```
a=b+c+d;
```

これを４バイト・３オペランド命令体系の命令列（命令列（Ａ１））に変換すると以下のようになる。

【００１９】

```
add    Rb, Rc, Ra
```

一方これを２バイト・２オペランド命令体系の命令列（命令列（Ａ２））に変換すると以下のようになる。

【００２０】

mov　Rb, Ra
add　Rc, Ra
add　Rd, Ra
４バイト・３オペランドの命令体系であれば、実行命令数は２であるが、命令メモリでの格納（および実行のための命令フェッチ）バイト数は８バイトである。一方２バイト・２オペランドの命令体系であると、実行命令数は３に増えるが、命令メモリでの格納（および実行のための命令フェッチ）バイト数は６バイトに減少する。この傾向は一般的に成立する。そして、４バイト・３オペランド命令体系は２バイト・２オペランド命令体系に較べ実行命令数が１～２割程度少ないが、格納バイト数は６割程度多くなる、という点が一般的に認められる。

【００２１】しかし、２バイト・２オペランドの命令体系には１つ課題が存在する。それは２オペランド命令体系の場合に必要となる余分なデータ転送命令にかかわる。上の式（Ａ）でも同様に説明できるのであるが、ここでは以下の式（Ｂ）で説明する。

【００２２】a=b+c;

これを４バイト・３オペランドの命令体系の命令列（命令列（Ｂ１））に変換すると以下のようになる。

【００２３】add　Rb, Rc, Ra

一方これを２バイト・２オペランドの命令列（命令列（Ｂ２））に変換すると以下のようになる。

【００２４】

mov　Rb, Ra
add　Rc, Ra

４バイト・３オペランドの命令体系であれば、パイプラインの片方だけを使用して１クロックで実行できる。一方２バイト・２オペランドの命令体系であれば、余分に必要となった複写（データ転送）命令mov と後続の加算命令addの２つの命令間にデータフローが存在する。つまり先行命令の結果の値を後続命令が使用している。したがって先行命令movの結果を待って後続命令addを実行する必要があり、２クロックの実行時間がかかる。以下の命令列

mov　Rb, Ra
add　Rc, Rd

であれば、２つの命令間でデータフローがないので、２つのパイプラインを使用して１クロックで実行できる訳であるが、式（Ｂ）に対応する命令列（Ｂ２）ではデータフローが存在することにより処理時間が余分にかかることになる。スーパスカラ方式を採用した場合、２バイト・２オペランド命令体系は４バイト・３オペランド命令体系に較べ、実行命令数の多さ以上に実行時間がかかる傾向があるといえる。

【００２５】なお、２バイト・２オペランド命令体系の課題を４バイト・３オペランド命令体系と比較して説明したが、４バイト・３オペランド命令体系においても、４オペランドの演算を実行する場合、前記命令列（Ａ１）のようにデータフローが存在し、２バイト・２オペランド命令体系と同様な課題が存在する。

【００２６】従来から存在するマイクロプロセッサは、ソフトウエア資産の蓄積があり、これまで築き上げてきたソフトウエア資産の継承の関係で、命令フォーマット、命令体系を変更することは困難である。従って、従来の命令フォーマット、命令体系を維持したまま、処理速度の向上を図る必要がある。

【００２７】本発明の課題は、スーパスカラ方式におけるデータ・ハザードのよるパイプライン・ストールを削減し、処理速度の向上を実現することにある。

【００２８】本発明の他の課題は、実行命令数を削減し、処理速度の向上を実現することにある。

【００２９】さらに、本発明の他の課題は、２バイト・２オペランド命令体系を実行するデータ処理装置の処理速度の向上を実現することにある。

【００３０】本発明の前記並びにその他の課題と新規な特徴は本明細書の記述及び添付図面から明らかになるであろう。

【００３１】

【課題を解決するための手段】本願において開示される発明のうち代表的なものの概要を簡単に説明すれば下記の通りである。

【００３２】パイプライン方式のデータ処理装置は、命令メモリに格納される固定長命令を読み込むステージと、読み込まれた複数の命令が実行するデータに依存性が有り、かつ前記複数の命令に所定の関係がある場合、前記複数の命令を複数のパイプラインで並列に実行できるように前記複数の命令を変更するステージと、変更された前記複数の命令を並列に実行するステージとを有する。

【００３３】命令体系上は２バイト２オペランド命令体系であるが、内部処理的には３オペランド命令体系として処理する。つまり、命令フェッチステージは２命令をフェッチする。命令デコードステージは２つの隣接した命令をデコードする。演算ステージの演算器は２組用意する。そして、隣接する２つの２オペランド命令が、１つの３オペランド命令と同等であることを検出する手段と、そうであれば２つの命令を１つの３オペランド命令に統合して後続の実行ステージに送出する手段を命令デコーダに設ける。これにより、１つの３オペランド命令として実行ステージに送られ１つのクロックで実行される。また隣接する２つの命令がデータフローの関係にあるが１つの３オペランド命令には統合できないことを検出すると、先行命令のソースデータを後続命令のための演算器に送る手段を設ける。

【００３４】これにより、２つの命令を同時に実行できることになる。以上の２つにより、隣接命令間のデータフローにより従来であれば２クロックの時間を要していた２つの命令処理を１クロックで実行できることにな

る。したがって、全体としての実行クロック数を削減できる。

【００３５】

【発明の実施の形態】本発明の実施例に係るマイクロプロセッサを順次項目に従って説明する。

【００３６】《マイクロプロセッサのパイプラインデータパス》図６には本発明の実施例に係るマイクロプロセッサのパイプラインのデータパスが示される。前記マイクロプロセッサは図５に示すような２バイト・２オペランド命令体系の命令をフェッチし、実行するものであるとして以下説明する。

【００３７】第１ステージ７００は命令フェッチステージである。第２ステージ８００は命令デコードステージである。第３ステージ９００は演算ステージである。第４ステージ１０００はレジスタへの書き込みとフォワーディングを行うステージである。前記各ステージの間には、第１ラッチ群７５０、第２ラッチ群８５０及び第３ラッチ群９５０がある。なお、図６以下の実施例における各ステージは、データの流れを示すもので、各ステージ内に記載される回路等の物理的な配置を示すものではない。

【００３８】《命令フェッチステージ》図７には第１ステージ７００と第１ラッチ群７５０との詳細ブロック図が示される。第１ステージ７００は、プログラムカウンタ（ＰＣ）７０１とフェッチ制御部７０２と命令メモリ７０３とで構成される。第１ステージ７００の命令フェッチステージの役割は命令メモリ内の命令を次の第２ステージ８００の命令デコードステージに渡すことである。

【００３９】プログラムカウンタ７０１の指すアドレスを信号線７０４に送出し命令メモリ７０３内の命令４バイト（２命令）を信号線７０５を介してフェッチ制御部７０２にフェッチする。フェッチ制御部７０２にフェッチされた２つの命令を信号線８０３に従って、信号線７０６、７０７に送出する。それから第１ラッチ群７５０内のラッチ７５１に信号線７０６の内容が格納され、ラッチ７５２に信号線７０７の内容が格納される。ラッチ７５１には第１命令が、ラッチ７５２には第２命令が格納される。ここで、命令列の中において第１命令は第２命令よりも先にある。なお、本願では第１命令を先行命令、第２命令を後続命令ともいう。

【００４０】また、プログラムカウンタ７０１の値に４を加えた値をプログラムカウンタ７０１に設定しなおす。プログラムカウンタ７０１の値（命令メモリをアクセスするアドレスの値）は２の倍数という制約のもとで命令メモリから４バイト分の命令（２命令）をフェッチし第１ラッチ群７５０内にラッチするよう第１ステージ７００が動作する。但し、常に命令メモリからフェッチした４バイト分の命令をそのまま第１ラッチ群７５０にラッチするものではない。すなわち、第２ステージ８００である命令デコーダステージから見て、次に欲しい命令が現命令の何バイト先かの情報を信号線８０３を介して第１ステージ７００のフェッチ制御部７０２に送る。第１ステージ７００のフェッチ制御部７０２はそれに応答してフェッチ制御部７０２内に存在するバッファを活用して命令デコードステージの望みの４バイト（２命令）を信号線７０６、７０７に送出し、第１ラッチ群７５０内のラッチ７５１、７５２に格納する。

【００４１】《命令デコードステージ》図８には第２ステージ８００と第２ラッチ群８５０との詳細ブロック図が示される。第２ステージ８００は、デコード制御部８０１とレジスタファイル８０２とで構成される。第２ステージ８００の命令デコードステージの役割は以下の通りである。

（１）２つの命令で使用される入力データを用意し、次の演算ステージ（第３ステージ９００）に渡す。

【００４２】（２）２つの命令間のデータフローを検査し、先行命令（第１命令）の実行結果を後続命令（第２命令）が使用していなければ、２つの命令処理を演算ステージに依頼する。

【００４３】（３）２つの命令間のデータフローを検査し、先行命令の実行結果を後続命令が使用していれば、所定の規則に従い２つの命令を変更する。

【００４４】（４）演算ステージに処理依頼した命令数を命令フェッチステージに連絡し、次のパイプラインの処理に備える。

【００４５】命令デコードステージ（第２ステージ８００）の動作を以下に説明する。　図１２にはデコード制御部８０１の一部の詳細ブロック図が示される。デコード制御部８０１はデータフロー検出回路ＤＦＤＣ、命令変換回路ＩＮＣＣ等を有する。命令変換回路ＩＮＣＣは、セレクタＳＥＬ１から４を有し、データフロー検出回路ＤＦＤＣの制御に基づいてラッチ７５１、７５２の内容を加工し、ラッチ８５１、８５２の内容に変換する。

【００４６】ラッチ７５１の内容である第１命令のＯＰフィールドをＯＰ－１、Ｄフィールドを Ｄ－１、Ｓ１フィールドをＳ１－１とする。ラッチ７５２の内容である第２命令のＯＰフィールドをＯＰ－２、Ｄフィールドを Ｄ－２、Ｓ１フィールドをＳ１－２とする。ラッチ８５１の内容である第１命令のＯＰフィールドをＯＰＮ－１、ＤフィールドをＤＮ－１、Ｓ１フィールドをＳ１Ｎ－１とする。ラッチ８５２の内容である第２命令のＯＰフィールドをＯＰＮ－２、ＤフィールドをＤＮ－２、Ｓ１フィールドをＳ１Ｎ－２とする。ラッチ８５２の内容である第２命令はさらにＳ２フィールドを有し、これをＳ２Ｎ－２とする。

【００４７】デコード制御部８０１はラッチ群７５０内のラッチ７５１、７５２より先行命令と後続命令の２つの命令を信号線７５３、７５４を介して取り込む。そし

て先行命令のＤフィールド（Ｄ－１）のレジスタ番号が後続命令のＳ１フィールド（Ｓ１－２）又はＤフィールド（Ｄ－２）のレジスタ番号と等しいか否かをデータフロー検出回路ＤＦＤＣで検査する。

【００４８】レジスタ番号が等しくない場合、データフローは存在しないと判断できる。レジスタ番号が等しい場合、データフローが存在すると判断できる。そうすると、データフロー検出回路ＤＦＤＣは、制御信号８２１から８２４を出力し、セレクタＳＥＬ１から４をそれぞれ切り替えて信号線８１３、８０４を介して、ラッチ８５１、８５２に変換した第１命令、第２命令を格納する。なお、セレクタＳＥＬ１、 ＳＥＬ２の一つの入力にはＩＮＣＣで生成された無効命令ＮＯＰ８２０が常時入力される。

【００４９】さらに、 セレクタＳＥＬ２には、信号線８４０を介してデータフロー検出回路ＤＦＤＣにより生成した新たな命令が入力される。信号線８４０によりセレクタＳＥＬ２に入力される新たな命令は、データフロー検出回路ＤＦＤＣがラッチ７５１のＯＰ－１とラッチ７５２のＯＰ－２に基づいて生成したものであり、ラッチ８５２のＯＰ－２に格納される。生成される新たな命令の一例としては、ＯＰ－１が１ビットシフト命令asllでＯＰ－２が加算命令addのときに生成される１ビットシフト加算命令asl1addがある。

【００５０】セレクタＳＥＬ３は、Ｓ１－１またはＤ－２の一方の値を選択し、Ｓ１Ｎ－２に格納するためのものである。

【００５１】セレクタＳＥＬ４は、Ｓ１－１またはＳ１－２の一方の値を選択し、Ｓ２Ｎ－２に格納するためのものである。

【００５２】図１１には命令デコードステージの２つの命令を演算ステージの２つの命令に変換する規則(条件と演算ステージに渡る命令)が示されている。第１命令は、無効命令nopに変換されるか又は変換されないかのどちらかである。第２命令は命令形式を図５の２バイト・２オペランド形式ものから図４の４バイト・３オペランド形式ものに変換されるか又は無効命令nopに変換される。図１１のALUは算術演算（加算、減算等）や論理演算（論理積、論理和等）などの２入力演算命令を総称する命令名である。前述したように、zextALU は演算器への第１入力を０拡張し、そしてALU演算する命令である。asl1ALUは演算器への第１入力を１ビット左シフトし、そしてALU演算する命令である。

【００５３】図１１の（1）は２オペランド形式の演算命令で３オペランドの演算命令を実行するためには複写命令movと演算命令ALUとの２命令必要であったものを１つの３オペランドの演算命令ALUに変換するものである。複写命令movのＤフィールドのレジスタ番号と演算命令ALUのＤフィールドのレジスタ番号とが一致する場合である。この場合、演算ステージには第１命令が無効命令nopに、第２命令が３オペランドの演算命令に変換されて渡される。

【００５４】ラッチ８５１，８５２の各フィールドに格納される値を要約すると以下のようになる。なお、「←」は、「←」の右側の値を「←」の左側に格納することを意味する。

【００５５】

ＩＦ　(D－1)＝(D－2)、
　THEN　OPN－1←nop、
　　OPN－2←OP－2、
　　DN－2←D－2、
　　S1N－2←S1－1、
　　S2N－2←S1－2

具体的には以下のようになる。ラッチ７５１のＯＰ－１には「mov」が、Ｄ－１には「ＲＮ」が、Ｓ１－１には「Ｒm」が格納されているとする。また、ラッチ７５２のＯＰ－２には「ALU」が、Ｄ－２には「ＲＮ」が、Ｓ１－２には「Ｒ１」が格納されているとする。ここでＤ－１とＤ－２が共に「ＲＮ」でレジスタ番号が一致することをデータフロー検出回路ＤＦＤＣが検出する。するとデータフロー検出回路ＤＦＤＣは、ＳＥＬ１がnop命令８２０を選択するように８２１を介しセレクタＳＥＬ１を制御し、nop命令８２０をラッチ８５１のＯＰＮ－１に格納する。 データフロー検出回路ＤＦＤＣは、ラッチ７５１のＤ－１、 Ｓ１－１をそのまま信号線７５３、８１３を介してラッチ８５１のＤＮ－１、 Ｓ１Ｎ－１に格納する。

【００５６】またデータフロー検出回路ＤＦＤＣは、セレクタＳＥＬ２がラッチ７５２のＯＰ－２を選択するように制御信号８２２を介してセレクタＳＥＬ２を制御し、ラッチ７５２のＯＰ－２をラッチ８５２のＯＰＮ－２に格納する。さらにデータフロー検出回路ＤＦＤＣは、セレクタＳＥＬ３がラッチ７５１のＳ１－１を選択するように制御信号８２３を介しセレクタＳＥＬ３を制御し、ラッチ７５１のＳ１－１をラッチ８５２のＳ１Ｎ－２に格納する。またデータフロー検出回路ＤＦＤＣはラッチ７５２のＤ－２を信号線７５４を介してそのままラッチ８５２のＤＮ－２に格納する。さらにデータフロー検出回路ＤＦＤＣは、セレクタＳＥＬ４がラッチ７５２のＳ１－１を選択するように８３４を介してセレクタＳＥＬ４を制御し、ラッチ７５２のＳ１－１をＳ２Ｎ－２に格納する。

【００５７】図１１の（2）は複写命令movのDフィールドのレジスタ番号と演算命令ALUのＳ１フィールドのレジスタ番号とが一致する場合である。この場合、演算ステージには第１命令はそのままで、第２命令が３オペランドの演算命令に変換されて渡される。

【００５８】ラッチ８５１，８５２の各フィールドに格納される値を要約すると以下のようになる。

IF　(D－1)＝(S1－2)、
　THEN　OPN－1←OP－1、
　　DN－1←D－1、
　　S1N－1←S1－1、
　　OPN－2←OP－2、
　　DN－2←D－2、
　　S1N－2←S1－1、
　　S2N－2←D－2

具体的には以下のようである。ラッチ７５１のＯＰ－１には「mov」が、Ｄ－１には「ＲＮ」が、Ｓ１－１には「Ｒｍ」が格納されているとする。また、ラッチ７５２のＯＰ－２には「ALU」が、Ｄ－２には「Ｒｘ」が、Ｓ１－２には「ＲＮ」が格納されているとする。ここでＤ－１とＳ１－２が共に「ＲＮ」でレジスタ番号が一致することをデータフロー検出回路ＤＦＤＣが検出する。そしてデータフロー検出回路ＤＦＤＣは、セレクタＳＥＬ１がラッチ７５１のＯＰ－１(この場合mov命令)を選択するように８２１を介しセレクタＳＥＬ１を制御し、mov命令をラッチ８５１のＯＰＮ－１に格納する。

【００５９】データフロー検出回路ＤＦＤＣは、ラッチ７５１のＤ－１、Ｓ１－１を、そのまま信号線７５３、８１３を介してラッチ８５１のDN－１、Ｓ１Ｎ－１に格納する。またデータフロー検出回路ＤＦＤＣは、セレクタＳＥＬ２がラッチ７５２のＯＰ－２を選択するように制御信号８２２を介してセレクタＳＥＬ２を制御し、ラッチ７５２のＯＰ－２をラッチ８５２のＯＰＮ－２に格納する。なお　データフロー検出回路ＤＦＤＣは、ラッチ７５２のＤ－２をそのまま信号線７５４、８０４を介してラッチ８５２のＤＮ－２に格納する。さらにデータフロー検出回路ＤＦＤＣは、セレクタＳＥＬ３がラッチ７５１のＳ１－１を選択するように制御信号８２３を介してセレクタＳＥＬ３を制御し、信号線８０４を介してラッチ８５２のＳ１Ｎ－２にラッチ７５１のＳ１－１を格納する。なおデータフロー検出回路ＤＦＤＣは、ラッチ７５２のＳ１－２を信号線７５４、８０４を介して、そのままラッチ８５２のＳ２Ｎ－２に格納する。

【００６０】なお、ラッチ８５１、８５２に具体的に格納される値を作っていく以上のような説明は図１１の(2)以降では省略する。図１１の(1),(2)と同様なやり方でラッチ８５１,８５２に格納する値を作っていけるからである。

【００６１】図１１の（3）は１オペランド形式の１ビット左シフト命令を２オペランド形式の１ビット左シフト命令に変換するものである。複写命令movの Ｄフィールドのレジスタ番号と１ビット左シフト命令asl1の Ｄフィールドのレジスタ番号とが一致する場合である。この場合、演算ステージには第１命令が無効命令nopに、第２命令が２オペランドの１ビット左シフト命令asl1に変換されて渡される。

【００６２】すなわち、各フィールドは下記のように変換される。

【００６３】

IF　(D－1)＝(S1－2)、
　THEN　OPN－1←nop、
　　OPN－2←OP－2、
　　DN－2←D－2orD－1、
　　S1N－2←S1－1、
　　S2N－2←NA

図１１の（4）は第１命令が複写命令movで、第２命令又は条件が図１１の（1）、（2）、（3）に該当しなかった場合である。この場合、演算ステージには第１命令はそのままで、第２命令が無効命令nopに変換されて渡される。「その他」の命令は１クロックずれた次のパイプラインで実行される。

【００６４】すなわち、各フィールドは下記のように変換される。

【００６５】

ＯＰＮ－１←ＯＰ－１、
ＤＮ－１←Ｄ－１、
Ｓ１Ｎ－１←Ｓ１－１、
ＯＰＮ－２←nop

図１１の（5）は０拡張命令zextと演算命令ALUとを０拡張演算命令zextALUに複合するものである。０拡張命令zextのＤフィールドのレジスタ番号と演算命令ALUのＤフィールドのレジスタ番号とが一致する場合である。この場合、演算ステージには第１命令が無効命令nopに、第２命令が３オペランドの０拡張演算命令zextALUに変換されて渡される。

【００６６】すなわち、各フィールドは下記のように変換される。

【００６７】

IF　(D－1)＝(D－2)、
　THEN　OPN－1←nop、
　　OPN－2←zexALU、
　　DN－2←D－2or D－1、
　　S1N－2←S1－1、
　　S2N－2←S1－2

図１１の（6）は０拡張命令zextの Ｄフィールドのレジスタ番号と加算命令addのＳ１フィールドのレジスタ番号とが一致する場合である。この場合、演算ステージには第１命令はそのままで、第２命令が３オペランドの０拡張加算命令zextaddに変換されて渡される。

【００６８】すなわち、各フィールドは下記のように変換される。

【００６９】

IF　(D－1) ＝ (S1－2)、
　THEN　OPN－1←OP－1、
　　DN－1←D－1、
　　S1N－1←S1－1、
　　OPN－2←zextadd、
　　DN－2←D－2、
　　S1N－2←S1－1、
　　S2N－2←D－2

なお、加算命令add以外に可換な論理積命令andや論理和命令or等も同様な変換を行っても良い。

【００７０】図１１の（７）は第１命令が０拡張命令zextで、第２命令又は条件が図１１の（５）又は（６）に該当しない場合である。この場合、演算ステージには第１命令はそのままで、第２命令が無効命令nopに変換されて渡される。「その他」の命令は１クロックずれた次のパイプラインで実行される。

【００７１】すなわち、各フィールドは下記のように変換される。

【００７２】

ＯＰＮ－１←ＯＰ－１、
ＤＮ－１←Ｄ－１、
Ｓ１Ｎ－１←Ｓ１－１、
ＯＰＮ－２←nop

図１１の（８）は１ビット左シフト命令asl1と演算命令ALUとを１ビット左シフト演算命令asl1ALUに複合するものである。１ビット左シフト命令asl1のＤフィールドのレジスタ番号と演算命令ALUのＤフィールドのレジスタ番号とが一致する場合である。この場合、演算ステージには第１命令が無効命令nopに、第２命令が３オペランドの１ビット左シフト演算命令asl1ALUに変換されて渡される。

【００７３】すなわち、各フィールドは下記のように変換される。

【００７４】

IF　(D－1) ＝ (D－2)、
　THEN　OPN－1←nop、
　　OPN－2←asl1ALU、
　　DN－2←D－2、
　　S1N－2←S1－1、
　　S2N－2←S1－2

図１１の（９）は１ビット左シフト命令asl1の Ｄフィールドのレジスタ番号と加算命令addのＳ１フィールドのレジスタ番号とが一致する場合である。この場合、演算ステージには第１命令はそのままで、第２命令が３オペランドの１ビット左シフト加算命令asl1addに変換されて渡される。

【００７５】すなわち、各フィールドは下記のように変換される。

【００７６】

IF　(D－1) ＝ (S1－2)、
　THEN　OPN－1←OP－1、
　　DN－1←D－1、
　　S1N－1←S1－1、
　　OPN－2←asl1add、
　　DN－2←D－2、
　　S1N－2←S1－1、
　　S2N－2←D－2

図１１の（１０）は第１命令が１ビット左シフト命令asl1で、第２命令又は条件が図１１の（８）又は（９）に該当しない場合である。この場合、演算ステージには第１命令はそのままで、第２命令が無効命令nopに変換されて渡される。「その他」の命令は１クロックずれた次のパイプラインで実行される。

【００７７】すなわち、各フィールドは下記のように変換される。

【００７８】

ＯＰＮ－１←ＯＰ－１、
ＤＮ－１←Ｄ－１、
Ｓ１Ｎ－１←Ｓ１－１、
ＯＰＮ－２←nop

図１１の（１１）は２つの命令間にデータフローがない場合のもので、命令の変換は行わない。

【００７９】デコード制御部８０１で変換された新しい２つの命令は信号線８１３、８０４に送出され、それぞれ第２ラッチ群８５０内のラッチ８５１、８５２に格納される。また、データフロー検出回路ＤＦＤＣにおける先行命令と後続命令との関係の検査結果を図１１のＰＣ更新の値に基づき命令フェッチステージ（第１ステージ７００）に信号線８０３を介して連絡する。すなわち、次のパイプラインでデコードする２つの命令を指定する情報を命令フェッチステージに連絡する。

【００８０】さらにデコード制御部８０１は先行命令のＳ１フィールド（Ｓ１－１）、Ｄフィールド（Ｄ－１）、さらに後続命令のＳ１フィールド５０３（Ｓ１－２）、Ｄフィールド５０２（Ｄ－２）の４つのレジスタ番号を信号線８０５、８０６、８０７、８０８を介してレジスタファイル８０２に送る。レジスタファイル８０２内の４つのレジスタの内容は、信号線８０９、８１０、８１１、８１２に読み出され、第２ラッチ群７４内のラッチ８５３（第１－１入力）、ラッチ８５４（第１－２入力）、ラッチ８５５（第２－１入力）、ラッチ８５６（第２－２入力）に格納される。

【００８１】図１５には、レジスタファイル８０２のブロック図が示される。レジスタファイル８０２は、レジスタＲＧＳＴＲとレジスタ制御回路ＲＣＣと等で構成される。レジスタＲＧＳＴＲは、４本のリードポートと２本のライトポートとがあり、それぞれ信号線８０９、８１０、８１１、８１２、信号線９５５、９５６に接続される。従って、レジスタファイル８０２は４つのレジス

タの内容を同時に読み出すことができる。また、２つのレジスタに同時に書き込むことができる。

【００８２】図１１の（１）、（５）、（８）の場合は、（Ｓ１－１）と（Ｓ１－２）で指定される２つのレジスタの内容が信号線８１１、８１２に読み出され、ラッチ８５５（第２－１入力）、ラッチ８５６（第２－２入力）に格納される。

【００８３】図１１の（２）、（６）、（９）の場合は、（Ｓ１－１）で指定されるレジスタの内容が信号線８０９、８１１に読み出され、ラッチ８５３（第１－１入力）とラッチ８５５（第２－１入力）とに格納される。（Ｄ－２）で指定されるレジスタ内容が信号線８１２に読み出され 、ラッチ８５６（第２－２入力）に格納される。

【００８４】図１１の（３）の場合は、（Ｓ１－１）で指定されるレジスタの内容が信号線８１１に読み出され、ラッチ８５５（第２－１入力）に格納される。

【００８５】図１１の（４）、（７）、（１０）の場合は、（Ｓ１－１）で指定されるレジスタの内容が信号線８０９に読み出され、ラッチ８５３（第１－１入力）に格納される。

【００８６】図１１の（１１）の場合は、（Ｓ１－１）、（Ｄ－１）、（Ｓ１－２）、（Ｄ－２）で指定される４つのレジスタの内容が信号線８０９、８１０、８１１、８１２に読み出され、ラッチ８５３（第１－１入力）、ラッチ８５４（第１－２入力）、ラッチ８５５（第２－１入力）、ラッチ８５６（第２－２入力）に格納される。

【００８７】《実行ステージ》図９には第３ステージ９００と第３ラッチ群９５０との詳細ブロック図が示される。第３ステージ９００は、演算制御部９０１とＡＬＵ(Alithmetic Logic Unit)等を含む演算器９０２、９０３と第１入力調整回路９０４、９０５、選択器９０６、９０７とで構成される。第３ステージ９００である実行ステージの役割は、２つの命令の演算を実行することである。

【００８８】演算器９０２と第１入力調整回路９０４は先行命令を演算するための回路で、第２ラッチ群８５０内の２つのラッチ８５３、８５４から第１－１入力、第１－２入力が信号線８５９、８６０を介して選択器９０６に送られる。また、第３ラッチ群９５０内の２つのラッチ９５３、９５４から第１出力、第２出力が信号線９５５、９５６を介して選択器９０６に送られる。

【００８９】選択器９０６は信号線８５９、９５５及び９５６のうちの１つを信号線１００１に従い選択して第１入力回路９０４及び信号線９１２を介して演算器９０２にデータを送る。また、選択器９０６は信号線８６０、９５５及び９５６のうちの１つを信号線１００１に従い選択して信号線９１３を介して演算器９０２にデータを送る。

【００９０】演算制御部９０１は第２ラッチ群８５０内のラッチ８５１の命令を取り込み、その命令機能に従い演算器９０２、第１入力調整回路９０４を信号線９１１と９０８で制御し、先行命令のための演算を行う。そして結果の値（第１出力）は第３ラッチ群９５０内のラッチ９５３に信号線９１８を介して格納される。

【００９１】一方、演算器９０３と第１入力調整回路９０５は後続命令を演算するための回路で、第２ラッチ群８５０内の２つのラッチ８５５、８５６から第２－１入力、第２－２入力が信号線８６１、８６２を介して選択器９０７に送られる。また、第３ラッチ群９５０内の２つのラッチ９５３、９５４から第１出力、第２出力が信号線９５５、９５６を介して選択器９０７に送られる。

【００９２】選択器９０７は信号線８６１、９５５及び９５６のうちの１つを信号線１００２に従い選択して第１入力回路９０５及び信号線９１４を介して演算器９０３にデータを送る。また、選択器９０７は信号線８６２、９５５及び９５６のうちの１つを信号線１００２に従い選択して信号線９１５を介して演算器９０３にデータを送る。演算制御部９０１は第２ラッチ群８５０内のラッチ８５２の命令を取り込み、その命令機能に従い演算器９０３、第１入力調整回路９０５を信号線９１０と９０９とで制御し、後続命令のための演算を行う。そして結果の値（第２出力）は第３ラッチ群９５０内のラッチ９５４に信号線９１９を介して格納される。

【００９３】以上が、実行ステージ（第３ステージ９００）の処理であるが、sladd命令やzextadd命令について補足説明しておく。aslladd命令やzextadd命令は、加算を実現できる演算器９０２または９０３への第１入力を微調整することで実現できる。すなわち第１入力を演算器に直接入力するのでなく、第１入力調整回路９０４または９０５に入力しそれを演算制御部９０１が制御し、１ビット左シフトや０拡張の調整を行っておき、それを演算器９０２または９０３へ入力し、そこで通常の加算をするよう制御することで実現できる。

【００９４】《書き込みステージ》図１０には第４ステージ１０００の動作を説明するためのブロック図が示される。第４ステージ１０００は、レジスタ番号解読回路１０１０とフォワーディング制御回路１０２０とで構成される。第４ステージ１０００であるレジスタへの書き込みとフォワーディングを行うステージの役割は以下の通り。

【００９５】（１）２つの命令の演算結果を指定された番号のレジスタに書き込む。

【００９６】（２）２つの命令の演算結果が現クロックでの演算ステージ（次のパイプライン）で使用されるなら第２ラッチ群８５０内にラッチされている値でなく、第３ラッチ群９５０内にラッチされている値を演算器に入力するように調整する（フォーワーディング）。

【００９７】まず、（１）の処理から説明する。第４ス

テージ１０００は、第３ラッチ群９５０内のラッチ９５１、９５２から直前に演算された２つの命令を信号線９５７、９５８を介してレジスタ番号解読回路１０１０に取り込む。また第３ラッチ群９５０内のラッチ９５３、９５４から直前の演算結果の値を信号線９５５、９５６に送出する。そして、レジスタ番号解読回路１０１０は直前に実行された命令の２つのＤフィールド内のレジスタ番号を信号線１００３、１００４に送出して第２ステージ８００のレジスタファイル８０２の書き込みレジスタ番号を指定する。これで２つの演算結果の値がレジスタファイル８０２に書き込まれることになる。

【００９８】次に（２）の処理を説明する。第４ステージ１０００は、第２ラッチ群８５０内のラッチ８５１、８５２から今回演算すべき２つの命令を信号線８５７、８５８を介してフォワーディング制御回路１０２０に取り込む。また、第３ラッチ群９５０内のラッチ９５１、９５２から直前に演算された２つの命令を信号線９５７、９５８を介してフォワーディング制御回路１０２０に取り込む。そして、フォワーディング制御回路１０２０は直前に実行された命令の２つのＤフィールド内のレジスタ番号と今回演算されるべき２つの命令のS１フィールド、S２フィールドの番号に同じものがあるか検査する。検査の結果同じものがあれば、その部分について、第２ラッチ群８５０内のラッチ８５３、８５４、８５５、８５６内の値でなく、第３ラッチ群９５０内のラッチ９５３、９５４内の値（信号線９５５、９５６）が演算器９０２、９０３に入力されるようにフォワーディング制御回路１０２０は信号線１００１、１００２を送出して２つの選択器９０６、９０７を制御する。

【００９９】《命令列の処理》図１３には本発明のスーパスカラ処理において命令列が個別のクロックでどのように処理されていくかが示されている。また、比較のため２命令が並列に実行できないときに無効命令nopを挿入するのみ場合に命令列が個別のクロックでどのように処理されていくかも示されている。本発明では、１クロック当たり２つの命令処理が可能となっている。また、本発明では、２命令が並列に実行できないときに無効命令nopを挿入する場合に比べて、実行命令数が６つ少なく実行時間が短くなる（この命令列においては約４０％実行命令が少なくなる）。

【０１００】先行命令がmov，zext，asll等の転送系命令で後続命令がaddなどの加算命令であれば、２つの命令を１つの命令に変換し、１つのクロックで実行するので、全体としてのクロック数を削減でき、高速化を図れる。また、先行命令が転送系命令で後続命令が演算命令であり、さらに両者間にデータフローが存在する場合でも、１つのクロックで実行するので、全体としてのクロック数を削減でき、高速化を図れる。

【０１０１】《マイクロコンピュータへの適用例》図１４には本発明のスーパスカラ方式を用いたマイクロプロコンピュータシステムが示される。マイクロコンピュータMCUは、中央処理装置ＣＰＵと、浮動小数点処理ユニットＦＰＵと、積和演算機能を有する乗算器ＭＵＬＴと、論理アドレスを物理アドレスに変換するメモリ管理ユニットMMUと、命令及びデータのキャッシュメモリＣＡＣＨＥと、キャッシュコントローラＣＣＮＴと、外部バスインタフェースＥＢＩＦと、３２ビット論理アドレスバスＬＡＢＵＳと、３２ビット物理アドレスデータバスＰＡＢＵＳと、３２ビットデータバスＤＢＵＳ、ＤＢＳとを単結晶シリコンのような半導体基板上に形成され、樹脂封止される（プラスチックパッケージに封止される）。

【０１０２】マイクロコンピュータMCUは、外部アドレスバスＥＡＢとデータバスＥＤＢを介してDRAM等のダイナミック記憶素子をメモリセルに使用した半導体メモリ等からなる主記憶装置MMに接続される。

【０１０３】中央処理装置ＣＰＵは、図６に示されるパイプライン・データパスで構成される。ただし、第３ステージと第４ステージとの間にメモリアクセスステージを有し、いわゆる５段パイプラインを構成する。なお、データメモリと命令メモリ７０３は、キャッシュメモリＣＡＣＨＥ又は主記憶装置MMに対応し、中央処理装置ＣＰＵ内には存在しない。中央処理装置ＣＰＵは２バイト固定長命令の命令体系の命令を実行し、演算器９０２、９０３は、３２ビット長のＡＬＵ等をそれぞれ有する。また、レジスタファイル８０２は、３２ビット長の汎用レジスタを１６本を有する。すなわち、中央処理装置ＣＰＵは特開平５－１９７５４６号公報に記載される２バイト・２オペランド命令体系（命令セット）の命令を実行する。特開平５－１９７５４６号公報に記載されるＣＰＵはスーパスカラ方式でない。これに比べて中央処理装置ＣＰＵはスーパスカラ方式であり、中央処理装置ＣＰＵは出願番号１９９２／８９７４５７号に記載される命令体系と同一の命令体系を実行できる。従って、従来のソフトウエアと互換性（オブジェクト・コード互換性）を維持しながら、高速性能を実現できる。また、２バイト固定長命令の特徴である高コード効率化も維持できる。

【０１０４】以上本発明者によってなされた発明を実施例に基づいて具体的に説明したが、本発明はそれに限定されるものではなく、その要旨を逸脱しない範囲において種々変更可能であることはいうまでもない。例えば、図６以下の実施例では、２バイト・２オペランド命令体系の場合について説明したが、４バイト・３オペランド命令体系の場合にも適用できる。０拡張命令、０拡張演算命令について説明したが、符号拡張命令、符号拡張演算命令についてもの同様に適用できる。また、第１命令の転送命令のＳ１フィールドはレジスタを指定する場合について説明したが、即値データである場合についても適用できる。

【０１０５】

【発明の効果】本願において開示される発明のうち代表的なものによって得られる効果を簡単に説明すれば下記の通りである。

【０１０６】隣接命令間のデータフローを検出し、命令を変換することにより、並列に命令を実行できる。従って、従来であれば複数クロックの時間を要していた複数の命令処理を１クロックで実行できる。それによって、全体としての実行クロック数を削減できる。

【図面の簡単な説明】

【図１】マイクロプロセッサのパイプライン化された実現方式を示す図。

【図２】パイプライン処理の概念を示す。

【図３】スーパースカラ処理の概念を示す。

【図４】４バイト命令体系の命令形式と命令レパートリの例を示す。

【図５】２バイト命令体系の命令形式と命令レパートリの例を示す。

【図６】本発明の実施例に係るマイクロプロセッサのパイプラインのデータパスを示す図。

【図７】第１ステージと第１ラッチ群との詳細ブロック図。

【図８】第２ステージと第２ラッチ群との詳細ブロック図。

【図９】第３ステージと第３ラッチ群との詳細ブロック図。

【図１０】第４ステージの動作を説明するブロック図。

【図１１】命令デコードステージの２つの命令を演算ステージの２つの命令に変換する規則を示す。

【図１２】デコード制御部の一部の詳細ブロック図を示す。

【図１３】命令列が個別のクロックでどのように処理されていくかを示す。

【図１４】本発明のスーパスカラ方式を用いたマイクロコンピュータシステムの図。

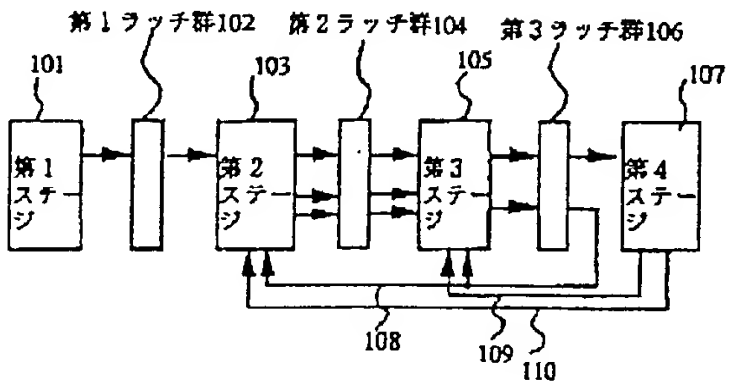
【図１５】レジスタファイルのブロック図。

【符号の説明】

１０１……第１ステージ、１０３……第２ステージ、１０５……第３ステージ、１０７……第４ステージ、１０８、１０９、１１０……信号線、４０１……ＯＰフィールド、４０２……Ｄフィールド、４０３……Ｓ１フィールド、４０４……Ｓ２フィールド、５０１……ＯＰフィールド、５０２……Ｄフィールド、５０３……Ｓ１フィールド、７００……第１ステージ、８００……第２ステージ、９００……第３ステージ、１０００……第４ステージ、７０１……プログラムカウンタ、７０２……フェッチ制御部、７０３……命令メモリ、７０４、７０５、７０６、７０７……信号線、７５１、７５２……ラッチ、８０１……デコード制御部、８０２……レジスタファイル、８０３、８０４、８０５、８０６、８０７、８０８、８０９、８１０、８１１、８１２、８１３……信号線、８５１、８５２、８５３、８５４、８５５、８５６……ラッチ、８５７、８５８、８５９、８６０、８６１、８６２……信号線、９０１……演算制御部、９０２……演算器、９０３……演算器、９０４……第１入力調整回路、９０５……第１入力調整回路、９０６……選択器、９０７……選択器、９０８、９０９、９１０、９１１、９１２、９１３、９１４、９１５、９１６、９１７、９１８、９１９……信号線、９５１、９５２、９５３、９５４……ラッチ、９５５、９５６、９５７、９５８……信号線、１００１、１００２、１００３、１００４……信号線、１０１０……レジスタ番号制御回路、１０２０……フォワーディング制御回路、ＩＮＣＣ……命令変換回路、ＤＦＤＣ……データフロー検出回路、ＭＣＵ……マイクロコンピュータ、　ＣＰＵ……中央処理装置、　ＦＰＵ……浮動小数点処理ユニット、　ＭＵＬＴ……乗算器、MMU……メモリ管理ユニット、ＣＡＣＨＥ……命令及びデータのキャッシュメモリ、　ＣＣＮＴ……キャッシュコントローラ、　ＥＢＩＦ……外部バスインタフェース、　ＬＡＢＵＳ……３２ビット論理アドレスバス、　ＰＡＢＵＳ……３２ビット物理アドレスデータバス、ＤＢＵＳ、ＤＢＳ……３２ビットデータバス、ＥＡＢ……外部アドレスバス、　ＥＤＢ……外部データバス、MM……主記憶装置、ＲＣＣ……レジスタ制御回路、ＲＧＳＴＲ……レジスタ。

【図１】

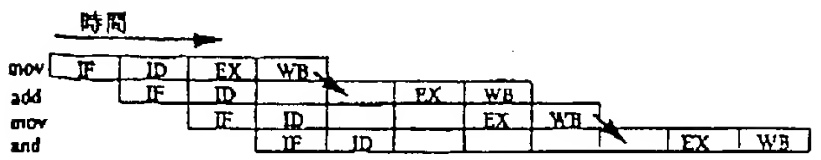
図１



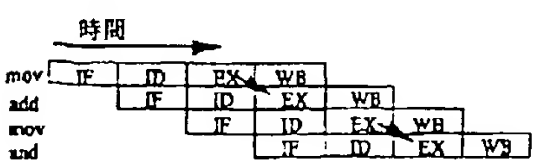
【図２】

図２

(a) データハザードによりパイプラインストールが発生した場合



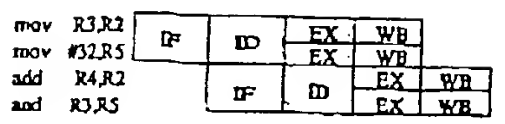
(b) フォワーデイングによりパイプラインストールがない場合



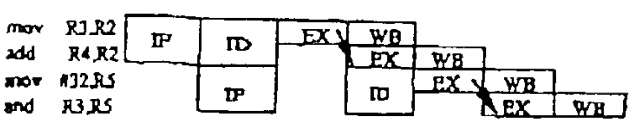
【図３】

図３

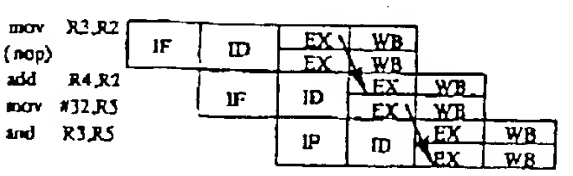
(a) データ依存無し



(b) データ依存有り

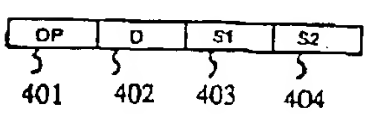


(C) データ依存有り（無効命令挿入）



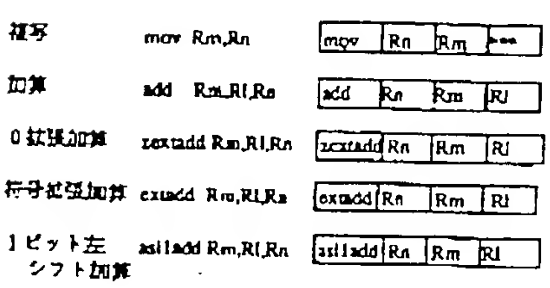
【図４】

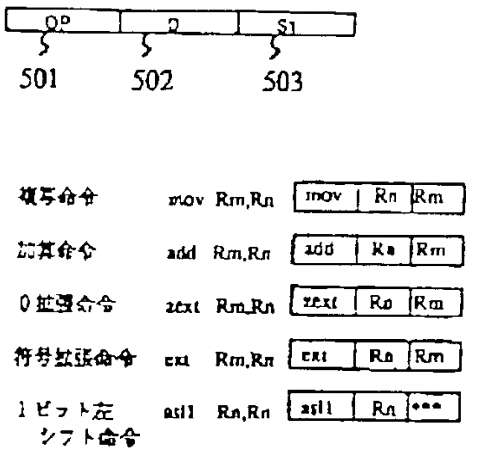
図４



【図５】

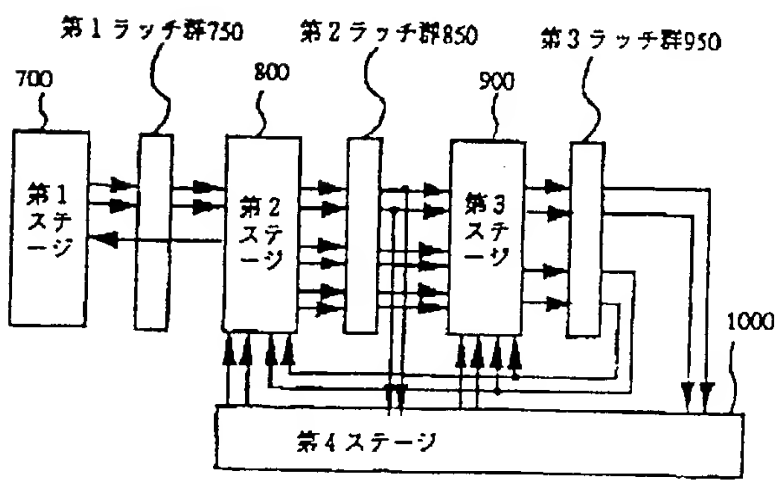
図５





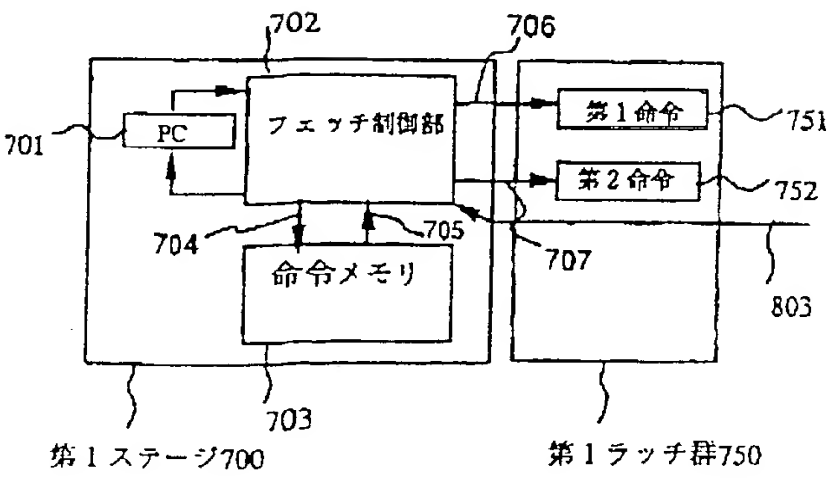
【図６】

図６



【図７】

図７



【図８】

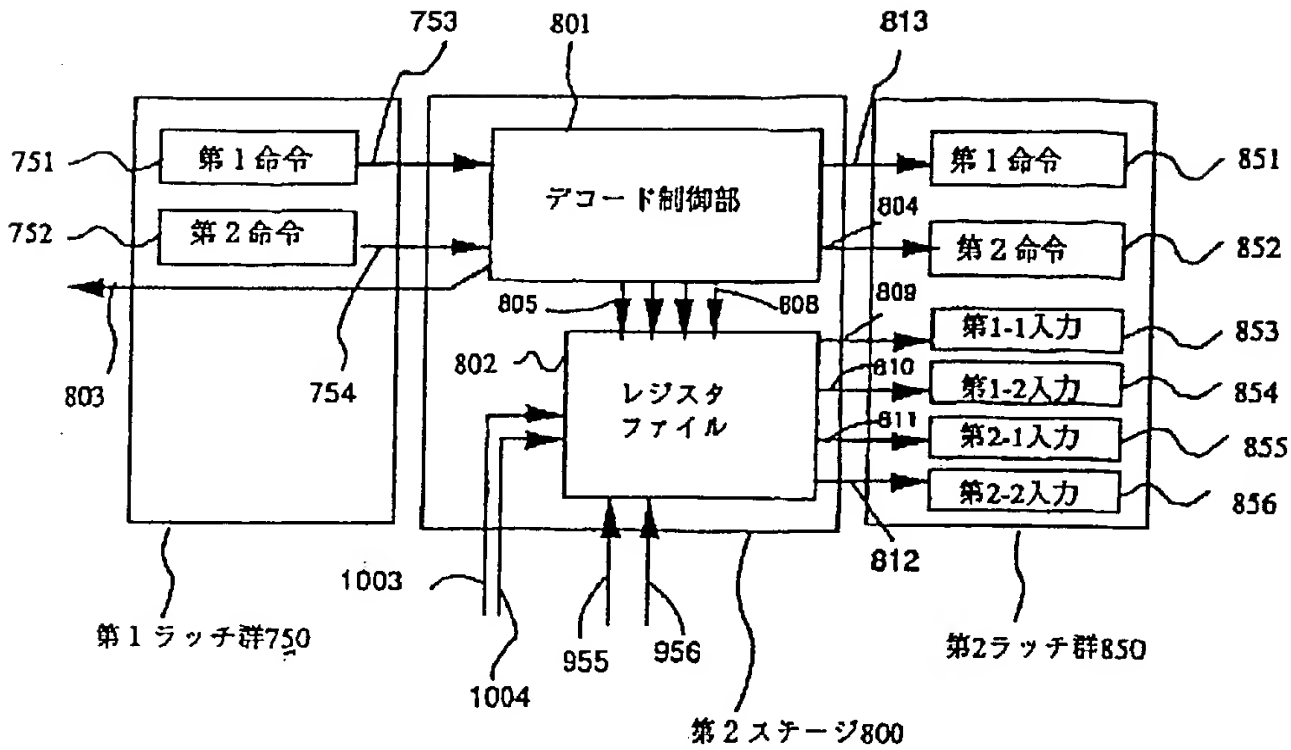

753
801
813
751
第１命令
752
第２命令
デコード制御部
804
第１命令
851
第２命令
852
805
808
809
第1-1入力
853
803
754
802
レジスタ
ファイル
810
第1-2入力
854
811
第2-1入力
855
第2-2入力
856
812
1003
第１ラッチ群750
955
956
第2ラッチ群850
1004
第２ステージ800

図８

【図９】

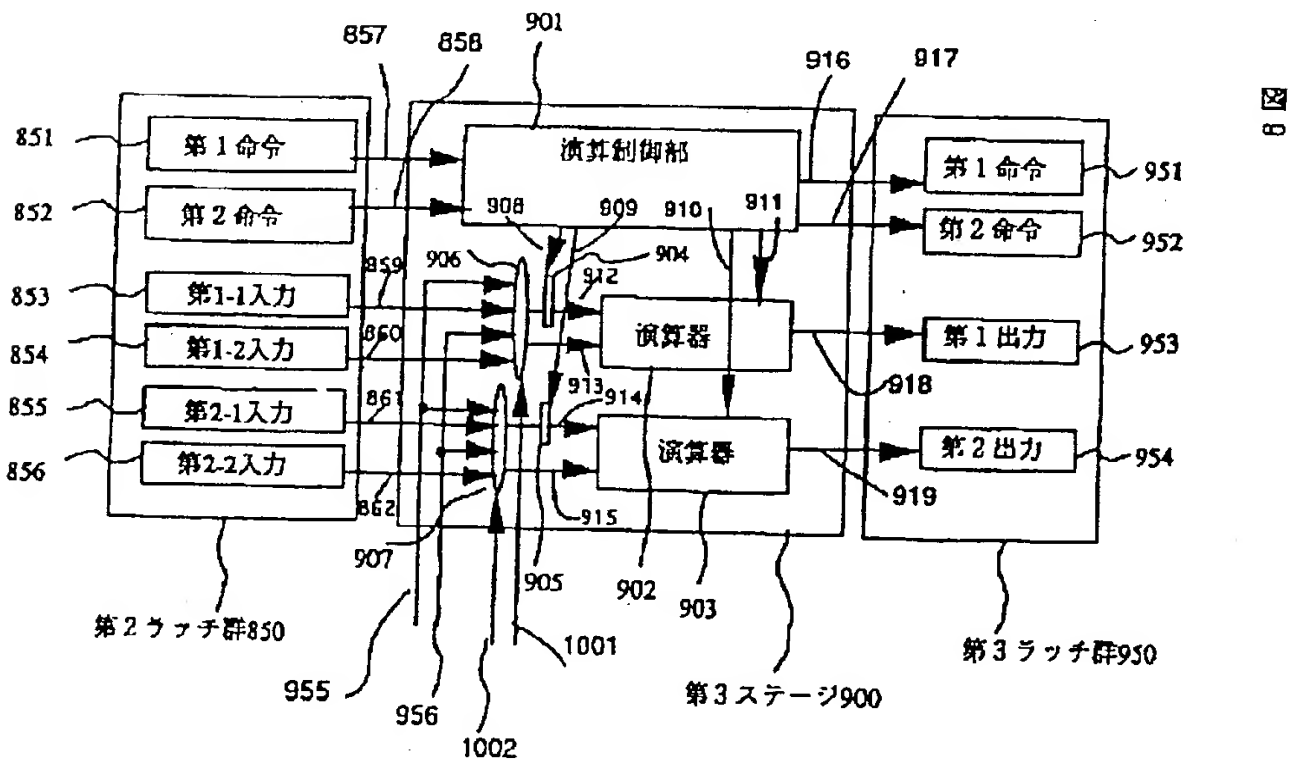

857
858
901
916
917
851
第１命令
演算制御部
第１命令
951
852
第２命令
908
909
910
911
第２命令
952
859
906
904
853
第1-1入力
912
演算器
第１出力
953
860
854
第1-2入力
913
914
918
861
855
第2-1入力
演算器
第２出力
954
856
第2-2入力
919
862
915
907
905
902
903
第２ラッチ群850
1001
955
956
第３ステージ900
第３ラッチ群950
1002

図９

【図１０】

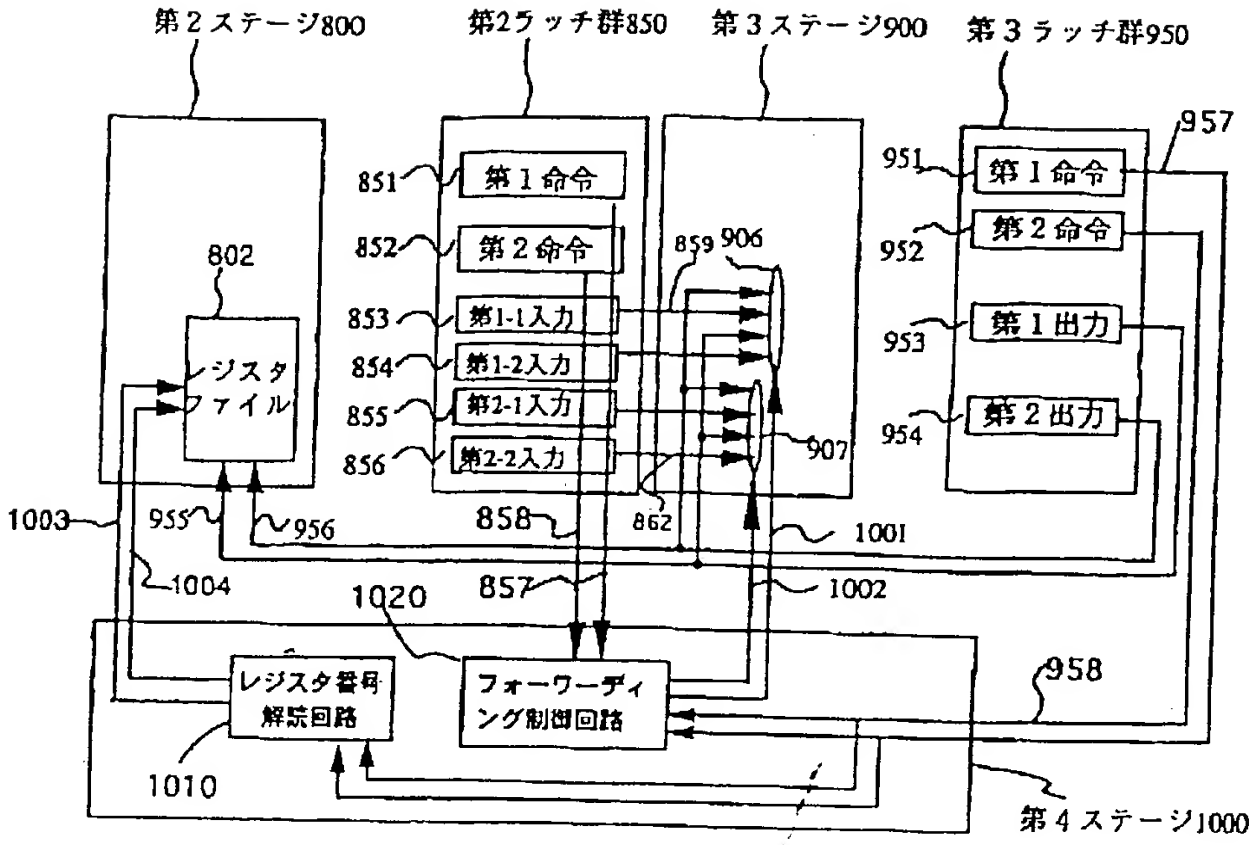


図１０

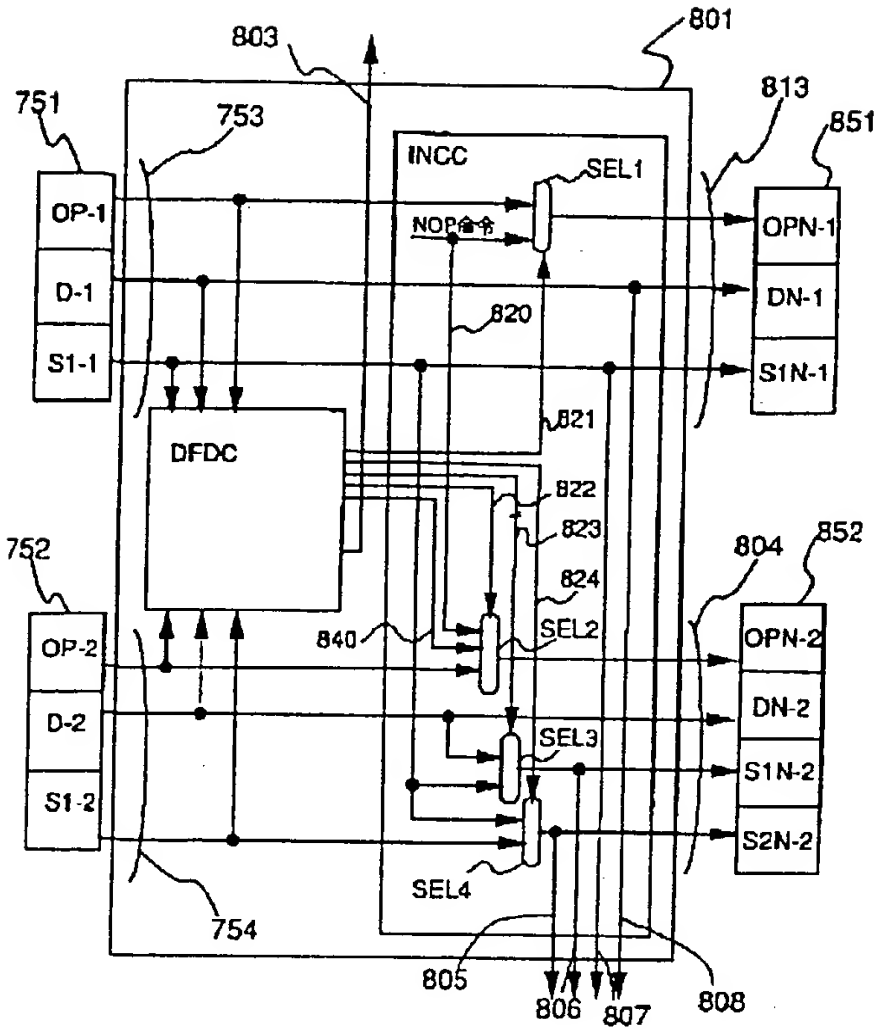
【図１１】

	デコードステージへの命令				演算ステージへの命令	
	第１命令	第２命令	条件	PCの更新	第１命令	第２命令
(1)	mov Rm,Rn	ALU Rl,Rx	n=x	+4	nop	ALU Rm,Rl,Rn
(2)	mov Rm,Rn	ALU Rl,Rx	n=l	+4	mov Rm,Rn	ALU Rx, Rm, Rx
(3)	mov Rm,Rn	asl1Rx,Rx	n=x	+4	nop	asl1 Rm,Rn
(4)	mov Rm,Rn	その他	---	+2	mov Rm,Rn	nop
(5)	zext Rm,Rn	ALU Rl,Rx	n=x	+4	nop	zextALU Rm,Rl,Rn
(6)	zext Rm,Rn	add Rl,Rx	n=l	+4	zext Rm,Rn	zextadd Rx,Rm,Rx
(7)	zext Rm,Rn	その他	---	+2	zext Rm,Rn	nop
(8)	asl1 Rn,Rn	ALU Rl,Rx	n=x	+4	nop	asl1ALU Rn,Rl,Rn
(9)	asl1 Rn,Rn	add Rl,Rx	n=l	+4	asl1 Rn,Rn	asl1add Rn,Rx,Rx
(10)	asl1 Rm,Rn	その他	---	+2	asl1 Rm,Rn	nop
(11)	ALU Rm,Rn	ALU Rl,Rx	n≠l n≠x	+4	ALU Rm,Rn	ALU Rl,Rx

図１１

【図１２】

図１２



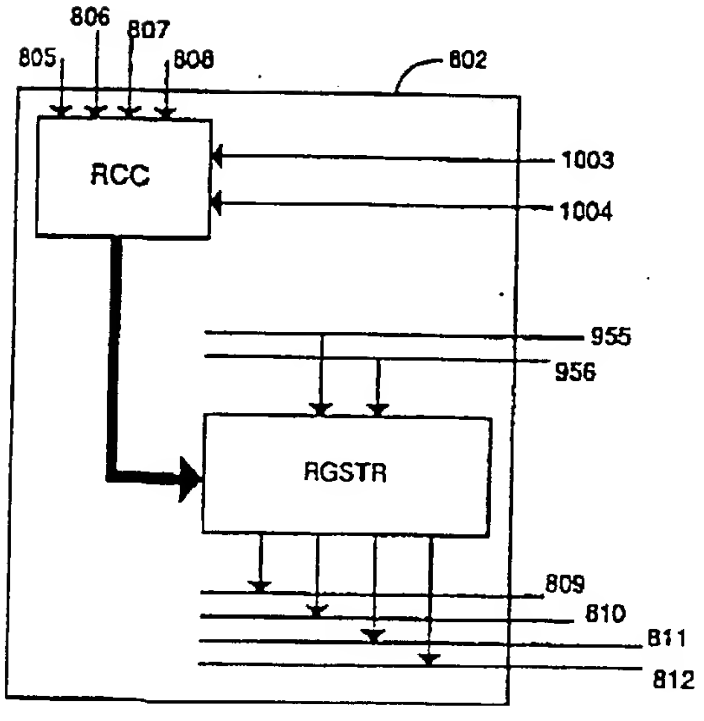
【図１３】

図１３

	クロック	第2ステージ８００への命令	第3ステージ９００への命令	nopを挿入した場合
1	mov Re,Rf			
	asl1 Rf,Rf			
2	zext Rb,Ra	nop		mov Re,Rf
	add Rc,Ra	asl1 Re,Rf		nop
3	mov Ra,Rb	nop		asl1 Rf,Rf
	add Rc,Rb	zextadd Rb,Rc,Ra		zext Rb,Ra
4	mov Rb,Rd	nop		add Rc,Ra
	add Rd,Re	add Ra,Rc,Rb		nop
5	add Rc,Rd	mov Rb,Rd		mov Ra,Rb
	...	add Rb,Re,Re		nop
6		add Rc,Rd		add Rc,Rb
		...		nop
7				mov Rb,Re
				nop
8				add Rd,Re
				nop
9				add Rc,Rd

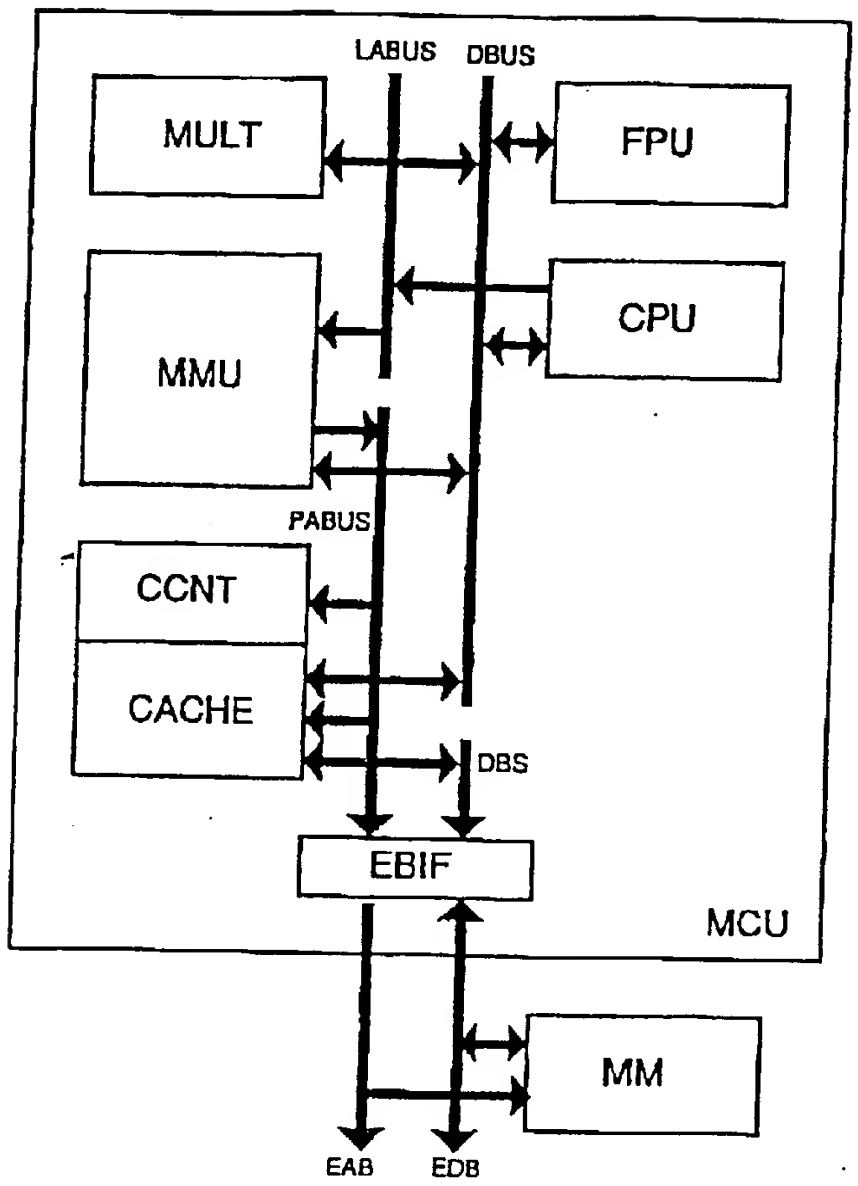
【図１５】

図１５



【図１４】

図１４


LABUS
DBUS
MULT
FPU
MMU
CPU
PABUS
CCNT
CACHE
DBS
EBIF
MCU
MM
EAB
EDB

Real Time Debug for System-on-Chip Devices

Authors:
Richard York, ASIC Designer
John Sharp, Development Systems Group
ARM Ltd., Cambridge, UK
June 1999

Debugging software running in a highly integrated system-on-chip (SOC) device at frequencies in excess of a 100MHz poses some significant engineering challenges. Traditional software debug tools such as in-circuit emulators (ICE) or logic analyzers have relied on having access to most of the signals of a microprocessor. This is not the case when the microprocessor core is deeply embedded in an SOC. In an extreme case, there may be no core signals visible on the pins of the chip. The lack of signals is not the only problem that traditional methods have difficulty overcoming. As frequencies exceed 100MHz, any additions to data path lengths can cause the skew of signals to such an extent that an incorrect representation of processor activity occurs.

The pin-out problem can be overcome by using bondout versions of the SOC, which provide all the signals needed. Bondouts take on one of two forms: either as an exact replica of the final SOC or as an implementation of a common subset of the functionality for a product range. Both have their problems. An exact replica is likely to be of use for only one product, therefore new bondouts are needed for each new project. A subset requires further logic to be added around the bondout to provide the functionality of the SOC, so it will most likely behave differently to the final chip. The use of bondout technology always adds delay and cost to the design cycle. The additional work required is technically challenging (difficulty of routing signals off chip at maximum frequency) thus diverting technical expertise away from the main objective.

There is a strong market need for the same functionality provided by traditional methods to be available for SOC solutions. The features provided by an ICE and supporting software can be broadly divided into five major categories:

1) Breakpointing and stepping code
2) An historical non-intrusive trace of instruction flow and data accesses
3) Non-intrusive access to registers and memory
4) Emulation memory
5) Code coverage and profiling

Breakpointing and stepping application code allows users to run the application code to a given point in code and then stop the processor. At this point the user has the option of examining or changing memory or register contents, stepping or restarting the application.

The really difficult bugs to track down are those that occur in situations where there is an unforeseen and hence unpredictable interaction between the application software and hardware. These bugs can be intermittent and usually only occur when the system is running at full speed; simply starting, stopping or stepping the processor does not expose the problem. An historical non-intrusive trace of instruction flow and data accesses can provide the extra information needed to identify the bug. For example, an application crashes during an interrupt routine. The result of the crash is that a memory protection fault occurs; the cause can not be found using breakpoints and single stepping methods. The user sets up the trace filter facility to collect trace data only during the interrupt routine, and the trigger to stop tracing when the protection fault occurs. The filter facility limits the amount of information that has been traced and analyzed. The trigger ensures that the trace information around the bug has been captured and not over written. As trace buffer depths are finite, these features are important to ensure the buffer is only filled

with relevant information. They also save time by limiting the information that needs to be analyzed to find the bug. Trigger and filter conditions can be changed to refine what trace data is captured and when.

Another key feature is the ability to change a processor's registers or memory without stopping the system. This is very useful in electro-mechanical applications. For example, when debugging an engine management system, the ability to change the control parameters of the engine without having to stop it will save time.

Many ICEs provide emulation memory that can be forcibly switched into a processor memory map. This is achieved by monitoring all processor accesses; when an access is made to a region that has been designated to be emulation memory, the access is re-routed. One of the major uses for this memory is the replacement of ROM during development, enabling the user to modify code that would normally be resident in ROM, allowing for quicker code development cycles.

The most recent addition to the debug toolset is the use of code coverage and analysis tools, which provide users with several useful benefits:

- Prove test coverage
- Reduce code size
- Improve code performance
- Provide minimum and maximum execution times for an algorithm

The code coverage and analysis tools are usually resident on the host controlling the ICE or logic analyzer; the information required is provided by the trace facility. These tools set the trace trigger and filter functions, and then use the captured data to provide the user with the relevant information.

All the above functions are features of current ICEs, which are now required in the SOC world if these systems are to be efficiently debugged. The ARM solution puts the real time components of in-circuit emulation into the SOC. The advantages of this approach are:

- Debug solution running at full processor speed
- Debug of final product
- Scalable solution for multiprocessor devices
- Standard tools for all ARM core-based SOCs
- Easy, reliable and small interconnect to target
- Low cost

We use three solutions to provide this functionality:

- EmbeddedICE logic
- Real Time Trace
- Real Time Monitor

The EmbeddedICE logic, which is an integral part of an ARM, contains breakpoint registers that compare the value on the core address, data and control busses against values programmed into the registers. For example, the logic may be programmed to generate a breakpoint when an instruction is loaded from a particular address or a particular data value is stored to a given location. When a breakpoint occurs the processor will be stopped and will then enter debug state. Once the core is in this state, memory and register contents can be examined or modified, images can be loaded, code can be stepped or execution restarted. The EmbeddedICE logic provides all of the standard run control debug features.

The Real Time Trace solution for ARM cores embedded within an SOC is made up of:

- The Embedded Trace Macrocell

- The Trace Port Analyzer
- The Trace Debug Tools

These three elements provide the capability to trace instructions and data accesses in real time. The Trace Macrocell monitors the ARM core busses and passes compressed information via the trace port to the Trace Port Analyzer. The Analyzer is an external device, which stores the information from the trace port. The trace information is compressed so that the analyzer does not need to capture data at the same bandwidth as an analyzer monitoring the core busses directly. This has the benefit of either lowering the cost or increasing the amount of processor activity that can be traced. The debug tools retrieve the data from the analyzer and reconstruct an historical view of the processor's activity, with data accesses to memory interleaved. The trace display includes full symbol information and links to the source code being debugged, allowing rapid understanding of the trace data. The trigger and filter logic, which is a part of the Embedded Trace Macrocell, is configured via the JTAG port.

The Real Time Monitor provides two major functions with minimal intrusion on the application execution time:

1) The debug of foreground tasks while interrupts continue
2) The ability to read and write memory without stopping the processor

With the EmbeddedICE logic described above, a breakpoint forced the processor into debug state and stopped the processor clock. However, with the recently announcedARM9E and ARM10 cores, enhanced EmbeddedICE-RT logic allows a breakpoint to either stop the processor or generate an exception. If an exception is generated, the processor is vectored to a small monitor program (less than 2KB code and data) resident in target memory that provides full debug functionality. Interrupts are still enabled, allowing application interrupt handlers to

run and commands from the debugger to be received via JTAG. These commands provide full debugging of the foreground task: it can be stepped or restarted, further breakpoints can be set, and memory read and written.

To read or write memory without stopping the processor, a similar mechanism is used. The user selects a memory location to be read or written. The debugger sends commands via JTAG to the monitor program, which momentarily interrupts the application. The monitor program carries out the required operation; control is then returned to the application.

In SOCs the common use of Flash memory and the execution of ROM code from faster on-chip RAM alleviate the need for emulation memory. Fast code download to the memory is necessary to reduce design cycles. Using JTAG and the EmbeddedICE logic, ARM's Multi-ICE product achieves download speeds of 120KB/s, hence a 1MB application can be downloaded to on-chip RAM in eight seconds. Multi-ICE can also reprogram on-chip Flash memory.

With the availability of analyzer buffer depths of up to 40 million instructions, ample trace information is provided by ARM's Real Time Trace solution for use by code coverage and analysis tools.

ARM's debug solution provides all the functionality of a traditional ICE for SOCs, with no external visibility of core signals running at frequencies well in excess of 100MHz. This solution is applicable to all ARM core-based designs available from any of the ARM semiconductor partners who incorporate the Embedded Trace Macrocell in their SOC devices.

###